# 1 Problem 1

#### 1.1 Code

```
import java.util.HashMap;
       CT255 Assignment 2
         This class provides functionality to build rainbow tables (with a different reduction function
         per round) for 8 character long strings, which consist of the symbols "a .. z", "A .. Z", "O .. 9", "!" and "#" (64 symbols in total).

Properly used, it creates the following value pairs (start value - end value) after 10,000 iterations of hashFunction() and reductionFunction():
6
                 start value - end value
                 Kermit12
                                    lsXcRAuN
                 Modulus!
                                    L2rEsY8h
10
                Pigtail1
                                    RONoLfOw
                 GalwayNo
                                    9PZjwF5c
11
                                    !oeHRZpK
12
                 Trumpets
                                    dkMPG7!U
                HelloPat
13
                pinky##!
                                    eDx58HRq
14
                 01!19!56
15
                 aaaaaaa
                                    rLtVvpQS
17
                 aaaaaaa
                                    kl06IeQJ
18
19
20
     * @author Michael Schukat
21
     * @version 1.0
22
23
24
    public class RainbowTable
25
         public static void main(String[] args) {
   long res = 0;
26
27
              // String array of the known passwords String[] passwords = {"Kermit12", "Modulus!", "Pigtail1", "GalwayNo", "Trumpets", "HelloPat", "pinky##!", "01!19!56", "aaaaaaaa", "aaaaaaaa"};
29
30
31
              HashMap<String, String> rainbowTable = new HashMap<>(); // declaring a HashTable that i'll
32
                   use to store the password : hash pairs
33
34
              // looping through the passwords array
35
              for (String start : passwords) {
   if (start.length() != 8) {
36
37
38
                        System.out.println("Input" + start + " must be 8 characters long - Exit");
40
                                                                                     // declaring a String hash that
41
                        String hash = start;
                             will hold the final reduced hash of a given password
42
                        // hashing & reducing the start String 10000 times. for (int i = 0; i < 10000; i++) {
43
45
                             hash = reductionFunction((hashFunction(hash)), i);
                        }
46
47
                        // adding the password & its hash value to the rainbowTable HashMap
48
                        rainbowTable.put(start, hash);
49
                   }
51
52
              // printing out the contents of the rainbowTable
53
              System.out.println(rainbowTable);
54
55
         private static long hashFunction(String s){
              long ret = 0;
58
              int i:
59
              long[] hashA = new long[]{1, 1, 1, 1};
60
              String filler, sIn;
61
62
              int DIV = 65536:
              filler = new String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");
65
66
              sIn = s + filler; // Add characters, now have "<input>HABCDEF..."
67
68
              sIn = sIn.substring(0, 64); // // Limit string to first 64 characters
              for (i = 0; i < sIn.length(); i++) {
70
                   char byPos = sIn.charAt(i); // get i'th character
71
```

```
72
                  hashA[0] += (byPos * 17111); // Note: A += B means A = A + B
                  hashA[1] += (hashA[0] + byPos * 31349);
hashA[2] += (hashA[1] - byPos * 101302);
hashA[3] += (byPos * 79001);
73
74
75
76
77
              ret = (hashA[0] + hashA[2]) + (hashA[1] * hashA[3]); if (ret < 0) ret *= -1;
78
79
80
              return ret;
81
82
         private static String reductionFunction(long val, int round) { // Note that for the first
              function call "round" has to be 0,
84
              String car, out;
                                                                                       // and has to be incremented by
                    one with every subsequent call.
                                                                                       // I.e. "round" created
              int i;
85
                   variations of the reduction function.
              char dat;
88
              \verb|car| = \verb|new| String("0123456789ABCDEFGHIJKLMNOPQRSTUNVXYZabcdefghijklmnopqrstuvwxyz!#"); \\
              out = new String("");
89
90
              for (i = 0; i < 8; i++) {
91
                  val -= round;
92
                   dat = (char) (val % 63);
93
                  val = val / 83;
out = out + car.charAt(dat);
95
96
97
             return out;
98
   }
```

## 1.2 Output

[andrewginspiron350] CT255-Assignment-2]\$ javac RainbowTable.java &6 java RainbowTable
[Kermit12=lsXcRAuN, GalwayNn=9R2]#F5c, aasaasaa=rLtVvpQs, HelloPat=dkMPG7!U, Modulus!=L2rEsY8h, Pigtail1=RONOLfOw, pinky##!=eDx58HRq, 01:19!56=vJ90ePjV, Trumpets=!oeHRZpK/
JandrewGinspiron350] CT255-Assignment-218

# 2 Problem 2

#### 2.1 Code

```
import java.util.HashMap;
    /* CT255 Assignment 2
     * This class provides functionality to build rainbow tables (with a different reduction function
        per round) for 8 character long strings, which consist of the symbols "a .. z", "A .. Z", "O .. 9", "!" and "#" (64 symbols in total).
5
        Properly used, it creates the following value pairs (start value - end value) after 10,000
6
             iterations of hashFunction() and reductionFunction():
7
               start value - end value
8
               Kermit12
                                  lsXcRAuN
                                  L2rEsY8h
9
               Modulus!
                                  RONoLf0w
               Pigtail1
10
               GalwayNo
                                  9PZjwF5c
11
                                  !oeHRZpK
               Trumpets
13
               HelloPat
                                  dkMPG7!U
                                  eDx58HRq
14
               pinky##!
               01!19!56
                                 vJ90ePiV
15
                                  rLtVvpQS
               aaaaaaa
16
17
               aaaaaaa
18
19
20
    * Qauthor Michael Schukat
21
     * @version 1.0
22
23
    public class RainbowTable
24
25
        public static void main(String[] args) {
   long res = 0;
26
27
28
             // String array of the known passwords
29
             The string [] passwords = {"Kermit12", "Modulus!", "Pigtail1", "GalwayNo", "Trumpets", "HelloPat", "pinky##!", "01!19!56", "aaaaaaaa", "aaaaaaaa"};
30
31
             HashMap<String, String> rainbowTable = new HashMap<>(); // declaring a HashTable that i'll
32
                  use to store the password : hash pairs
33
             // looping through the passwords array
             for (String start : passwords) {
37
                 if (start.length() != 8) {
```

```
38
                       System.out.println("Input " + start + " must be 8 characters long - Exit");
39
                  7
40
                  else {
                                                                               // declaring a String hash that
                       String hash = start:
41
                            will hold the final reduced hash of a given password
42
                       // hashing & reducing the start String 10000 times.
43
44
                       for (int i = 0; i < 10000; i++) {
45
                           hash = reductionFunction((hashFunction(hash)), i);
46
47
                       // adding the password & its hash value to the rainbowTable HashMap
48
                       rainbowTable.put(start, hash);
49
50
                  }
51
             }
              // printing out the contents of the rainbowTable
52
              System.out.println(rainbowTable);
53
54
              // chain lookup section
             // long array of the 4 hashes to be searched for long[] hashes = {895210601874431214L, 750105908431234638L, 1111111111115664932L,
56
57
                   977984261343652499L};
58
59
              // for each loop that loops through each hash in the array of hashes
             for (long hash : hashes) {
60
61
                  // looping 10000 times to search for the password - this will function as our \max
62
                       number of iterations, as 10000 iterations should just take use back to where we
                       started.
                  for (int i = 0; i < 10000; i++) {
63
                       // reducing the hash
                       String str = reductionFunction(hash, i);
                       // checking if the reduced hash is a key (password) in the rainbowTable HashMap
if (rainbowTable.containsValue(str)) {
66
67
                           System.out.println("Found password" + str + " for hash value" + hash);
                                                                                                                   //
68
                                printing the found password
                           break;
69
                                                                                                                   //
                                breaking out of the for loop
70
                       }
71
                       else {
                           hash = hashFunction(str):
                                                                                                                   11
72
                                hashing str before continuing the for loop
73
74
                  }
75
             }
76
77
         private static long hashFunction(String s){
78
             long ret = 0;
79
              int
             long[] hashA = new long[]{1, 1, 1, 1};
82
              String filler, sIn:
83
84
              int DIV = 65536;
85
             \texttt{filler} = \texttt{new} \; \texttt{String} ( \, "ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH"}) \, ; \\
87
88
              sIn = s + filler; // Add characters, now have "<input>HABCDEF..."
89
              sIn = sIn.substring(0, 64); // // Limit string to first 64 characters
90
91
              for (i = 0; i < sIn.length(); i++) {
92
                  char byPos = sIn.charAt(i); // get i'th character
hashA[0] += (byPos * 17111); // Note: A += B means A = A + B
hashA[1] += (hashA[0] + byPos * 31349);
hashA[2] += (hashA[1] - byPos * 101302);
94
95
96
                  hashA[3] += (byPos * 79001);
97
98
99
              ret = (hashA[0] + hashA[2]) + (hashA[1] * hashA[3]);
100
101
              if (ret < 0) ret *= -1;
             return ret:
102
103
104
         private static String reductionFunction(long val, int round) { // Note that for the first
              function call "round" has to be 0,
106
              String car, out;
                                                                                   // and has to be incremented by
                    one with every subsequent call.
                                                                                   // I.e. "round" created
              int i;
107
                  variations of the reduction function.
108
              char dat;
              {\tt car = new String("0123456789ABCDEFGHIJKLMNOPQRSTUNVXYZabcdefghijklmnopqrstuvwxyz!#");}
110
              out = new String("");
111
112
              for (i = 0; i < 8; i++) {
113
                  val -= round;
114
                  dat = (char) (val % 63);
115
                  val = val / 83;
116
                  out = out + car.charAt(dat);
117
118
119
```

```
120 return out;
121 }
122 }
```

#### 2.2 Output

I couldn't actually find a password match with the above code, and I'm not sure why. My current guess would be that the reduction function wasn't being called properly, as everything else *seemed* to be working as expected. I didn't call the reduction more than 10,000 times as that would theoretically just lead me back to the same place in the chain. I think that my problem is with the passing of the integer i to the reduction function, as I think that I correctly implemented the rest of the steps for performing a chain lookup - I input a hash value, reduce it, check if the reduced form is in the list of final plaintexts (the "Values" in the HashMap), and if so break out of the loop (but this never occurs), assigning the relevant "Key" from the HashMap as the original plaintext password that produced the original input hash. Otherwise, I continue until I'm back at the same place in the chain after the 10,000<sup>th</sup> iteration, where the code gives up.

[andrew@inspiron3501 CT255-Assignment-2]\$ javac RainbowTable.java && java RainbowTable
{Kermit12=lsXcRAuN, GalwayNo=9PZjwF5c, aaaaaaaa=rLtVvpQS, HelloPat=dkMPG7!U, Modulus!=L2rEsY8h, Pigtail1=RONOLfOw, pinky##!=eDx58HRq, 01!19!56=vJ90ePjV, Trumpets=!oeHRZpK}
[andrew@inspiron3501 CT255-Assignment-2]\$